PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Rocket Joe by Trader Joe

05 February 2022

paladinsec.co

info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 3

1 Overview 4

1.1 Summary 4

1.2 Contracts Assessed 4

1.3 Findings Summary 5

1.3.1 RocketJoeToken 6

1.3.2 LaunchEvent 6

1.3.3 RocketJoeStaking 7

1.3.4 RocketJoeFactory 7

2 Findings 8

2.1 RocketJoeToken 8

2.1.1 Token Overview 9

2.1.2 Privileges 9

2.1.3 Issues & Recommendations 10

2.2 LaunchEvent 12

2.2.1 Privileges 14

2.2.2 Issues & Recommendations 15

2.3 RocketJoeStaking 23

2.3.1 Privileges 23

2.3.2 Issues & Recommendations 24

2.4 RocketJoeFactory 30

2.4.1 Privileges 31

2.4.2 Issues & Recommendations 32

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Trader Joe's Rocket Joe contracts on the Avalanche network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Rocket Joe by Trader Joe
URL	https://traderjoexyz.com/
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
RocketJoeToken	0x5483ce08659fABF0277f9314868Cc4f78687BD08	MATCH
LaunchEvent	0x8E8C6bD1Ce1A77885325aa69F3736ED41d66532e	MATCH
RocketJoeStaking	0x3E6456b99bA6964f40b4f1c458A932fEB5677001	MATCH
RocketJoeFactory	0x00553fCA0428EDA1eb499337052636C229BeBF9a	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	1	-	1
Medium	0	-	-	-
Low	3	3	-	-
Informational	19	17	-	2
Total	**24**	**21**	**-**	**3**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 RocketJoeToken

ID	Severity	Summary	Status
01	INFO	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	ACKNOWLEDGED
02	INFO	Best practice: `_beforeTokenTransfer` does not call the parent `_beforeTokenTransfer`	RESOLVED

1.3.2 LaunchEvent

ID	Severity	Summary	Status
03	HIGH	An exploiter can prevent the liquidity addition step from ever happening by pre-initializing the pair with WAVAX	RESOLVED
04	LOW	LaunchEvent is a proxy but does not use upgradeable OpenZeppelin dependencies	RESOLVED
05	INFO	Best practice: Doing the user transfer before the fee transfer on withdrawals increases the exploit risk in case the contract is modified	RESOLVED
06	INFO	Typographical errors	RESOLVED
07	INFO	Lack of validation	RESOLVED
08	INFO	`withdrawAVAX` can be made external	RESOLVED
09	INFO	Lack of event for `withdrawIncentives`	RESOLVED
10	INFO	`getPenalty` reverts ambiguously if the launch has not started yet	RESOLVED

1.3.3 RocketJoeStaking

ID	Severity	Summary	Status
11	HIGH	Governance risk: The staking contract is upgradeable	ACKNOWLEDGED
12	INFO	Contract uses manipulatable contract token balance to figure out the total number of JOE staked	RESOLVED
13	INFO	Best practice: `deposit` does not adhere to checks-effects-interactions	RESOLVED
14	INFO	Check within `updatePool` that validates that nobody has staked yet can be circumvented by transferring a nominal JOE token to the contract leading to lost rewards	RESOLVED
15	INFO	Typographical errors	RESOLVED
16	INFO	Lack of validation on `updateEmissionRate` could cause excessive minting	RESOLVED

1.3.4 RocketJoeFactory

ID	Severity	Summary	Status
17	LOW	`eventImplementation` cannot be changed	RESOLVED
18	LOW	`safeTransferFrom` should be used	RESOLVED
19	INFO	`eventImplementation` and `wavax` can be made `immutable`	RESOLVED
20	INFO	Typographical errors	RESOLVED
21	INFO	`maxAllocation` is vulnerable to Sybil attacks	ACKNOWLEDGED
22	INFO	Setting rJoe back to a previous rJoe token will be impossible since it is already initialized	RESOLVED
23	INFO	Lack of validation	RESOLVED
24	INFO	Lack of events for `setPhaseDuration` and `setPhaseOneNoFeeDuration`	RESOLVED

2 Findings

2.1 RocketJoeToken

The Rocket Joe token is the primary token used by the Rocket Joe platform to participate in launch events. It must be burned proportionally to a user's participation size in a launch.

rJoe is earned over time by staking Joe tokens in the Staking contract. It is being emitted gradually while being burned whenever a launch event occurs. There is therefore no theoretical maximum supply but emissions and value can be kept in check with sufficient launch events.

rJoe does not have an initial supply and cannot be transferred by normal users. It can only be transferred and minted by the owner, which will be the Joe staking contract. It will be minted and transferred to users as a reward for staking Joe tokens. It can only be burned by launch events registered in the Rocket Joe factory.

2.1.1 Token Overview

Name	`0x5483ce08659fABF0277f9314868Cc4f78687BD08`
Symbol	rJoe
Address	TBD
Token Supply	Unlimited
Decimal Places	18
Transfer Max Size	No transfers permitted
Transfer Min Size	No transfers permitted
Transfer Fees	No transfers permitted
Pre-mints	None

2.1.2 Privileges

The following functions can be called by the owner:

- `mint`
- `burnFrom`
- `transferOwnership`
- `renounceOwnership`

2.1.3 Issues & Recommendations

Issue #01	**`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef**
Severity	INFORMATIONAL
Location	Line 37 `function mint(address _to, uint256 _amount) external onlyOwner {`
Description	The `mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the staking contract. This can be done while the token contract has been deployed but before ownership is transferred to the staking contract. The risk of this governance issue occurring in practice is especially prevalent amongst less-reputable projects, but as the client is one of the cornerstone projects of the Avalanche ecosystem, this issue has been reduced to informational severity in this report. Any pre-mints can be prominently seen on the Blockchain.
Recommendation	Consider being forthright if this `mint` function has been used by letting your community know how much was minted, where they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	ACKNOWLEDGED The client has indicated they will not do any pre-mint. This can be easily validated with the explorer and this issue can be marked as resolved on the request of the client once the contracts are deployed and ownership is transferred.

Issue #02	**Best practice: `_beforeTokenTransfer` does not call the parent `_beforeTokenTransfer`**
Severity	INFORMATIONAL
Location	Line 57 `function _beforeTokenTransfer(`
Description	The Rocket Joe token overrides the function that is called before any transfer to validate that the transfer is either a `mint`, `burn` or `transfer` from the staking contract. However, it no longer calls the parent `_beforeTokenTransfer` function. Although not important within the present deployment since the parent function is empty, it is generally considered best practice to still call the parent `_beforeTokenTransfer`. This will reduce the chances of error if ever an ERC20 extension that uses `beforeTokenTransfer` as well is implemented in a future fork.
Recommendation	Consider adding `super._beforeTokenTransfer(_from, _to, _amount);` at the end of the `_beforeTokenTransfer` override.
Resolution	RESOLVED

2.2 LaunchEvent

The LaunchEvent is the primary contract within the Rocket Joe protocol and is created for every token that is launched on Rocket Joe. It functions quite a bit differently than a traditional LaunchPad contract and notably aims to find the market price before liquidity is added. This is valuable as it avoids the type of frictions present in most of the current launchpads. One such friction is the sniping practice of the initial supply, often bringing in millions of dollars per launch to the sniping party. Such practices are avoided if the liquidity can be added at the "right price".

While creating a Launch Event, the project determines how many tokens they want to offer and transfers these in during event creation. They can also configure other parameters like the withdrawal penalty size. The project can set a floor price of AVAX per token which reduces the number of project tokens used to create the LP in case not enough AVAX was raised. In this case, if the market believes that the price should be lower than the floor price (eg. there is simply not enough interest), the floor price then acts as an insurance for the project to still add liquidity at the non-equilibrium price minimum chosen by the project. As long as enough AVAX tokens were raised to exceed this floor price, the whole reserved token allocation will be used to create the LP pair with all raised AVAX. Throughout the raising round, users can deposit and withdraw AVAX with the final price being based on how much AVAX is in the contract — this way, if users believe too much AVAX has been deposited and the price is too high, they can withdraw their stake again.

Events consist of three phases. The first phase is where deposits and withdrawals are both permitted. This first phase is split up in a period where withdrawals are free and a period where they have a linearly increasing withdrawal fee. Within the second phase, only withdrawals are permitted against a fixed withdrawal fee. The final phase is the liquidity addition and token withdrawal phase.

During the first phase of the event, people are free to deposit and withdraw AVAX in and out of the contract. When depositing for the first time, an rJoe amount proportional to the deposit size needs to be burned. When withdrawing, there is an initial period where withdrawals are free. After this period has expired, a linearly increasing withdrawal fee starts accumulating and withdrawals are no longer free. This means that early withdrawals after the free period still have a minor fee while the longer users wait to withdraw, the higher the fee. Deposits are still permitted at this point. This mechanism is used to discover the price optimum to some extent before having to add liquidity. All withdrawal fees are sent to the Trader Joe governance "penalty collector" address. They are therefore not directly given to the Launch Event project.

During the second phase, deposits are no longer permitted. However, users can still incur a fixed withdrawal fee and withdraw their AVAX. This way, if users believe too many people have deposited, they can withdraw their stake. It should be noted that speaking from a game-theory perspective, users will likely wait until the very last second to make their withdrawal. This is why the previous phase should be configured to always have a lower withdrawal fee to incentivize withdrawing early.

After phase two is finished, all AVAX in the contract is paired with the offering tokens and the LP is formed. Half of this LP will be given to the project while the other half is given to the users. The project and users need to wait for a timeframe configured by the project before they can withdraw the added liquidity. This vesting period is configured separately for the user and project itself and the vesting schedule is a simple single vesting time. This schedule means that all users will unlock their complete LP amount at a predetermined time with the project unlocking their complete half at another predetermined time.

Finally, the contract contains a separate feature from the LP token distribution mechanism called the incentive tokens. The project can determine a percentage of the offering amount that will immediately be vested to users when liquidity is added. This percentage is given in offering tokens and not LP tokens. Users will therefore be immediately able to make a sell decision if they want to use these

tokens. However, since they still have vesting LP tokens, this might not be in their aggregate interest, which is an interesting game-theoretical implication.

The withdrawal penalties can be set up to 50%. The user LP withdrawal lock can be set by the project to at most 7 days after liquidity has been added. The project LP withdrawal lock must always be longer than the user lock and there is furthermore no time limit on how long the project their LP half is locked. The Launch Event creator is not permitted to participate in the contract as this would cause malfunction with some of the business logic.

Although the user can supply and withdraw AVAX throughout the lifecycle of the event, they only need to burn rJoe for any given allocation once. This means that if the user supplies 1 AVAX, withdraws it and then resupplies it again, they do not need to burn rJoe a second time.

Finally, the Trader Joe governance can stop a launch event at any time by calling the `allowEmergencyWithdraw` function on this event. This allows participants to withdraw their deposits. The issuer can withdraw all reserve and incentives launch tokens after the Trader Joe governance enables emergency withdrawals.

2.2.1 Privileges

The following functions are privileged functions:

- `withdrawLiquidity`
- `allowEmergencyWithdraw [ Trader Joe ]`

2.2.2 Issues & Recommendations

Issue #03	**An exploiter can prevent the liquidity addition step from ever happening by pre-initializing the pair with WAVAX**
Severity	HIGH SEVERITY
Location	Line 410 `(, , lpSupply) = router.addLiquidity(`

Description	At the start of the third phase, the raised WAVAX is paired with the offering tokens and is added as liquidity to a newly created Trader Joe pair. However, as the client was already aware of at the time of our audit, a malicious party can pre-initialize this pair. They have correctly adjusted their business logic to account for the fact that an empty pair can potentially already exist.
	
	However, presently the code will revert in the edge case where an exploiter sends WAVAX to the pre-initialized LP pair and then calls `sync()` on the pair. This happens because a function within the Uniswap router, `quote`, can revert in this situation.
	
	As this was simply an hypothesis of ours at the time of raising the issue, we have worked together with the Rocket Joe team to write a test-case which they delivered to us promptly and genuinely appreciate their quick cooperation in investigating this. This test-case is summarized below in pseudo-code as a reference of how a malicious party would block any Launch Event from ever completing.
	
	`pair = factory.createPair(wavax.address, token.address);`
	`wavax.transfer(pair.address, 1 ether);`
	`pair.sync();`
	`launchEvent.createPair();`

This issue has been marked as high severity due to the fact that a malicious party can completely prevent any launch event from ever completing, but more importantly because at this point there is no way for Trader Joe to fix a bug like this for future LaunchEvents, as they would have to redeploy the whole Rocket Joe token to make modifications to the LaunchEvent contract. We have recommended setting new implementations in another issue to allow Rocket Joe to improve their launch event contract over time and add new features to it.

Finally it should be noted that this issue is in fact a flawed implementation of the Uniswap V2 router, as we believe a correctly written `addLiquidity` function should not revert in this scenario. We hope that Uniswap considers creating a UniswapV2Router03 to address the fact that `addLiquidity` does not work in this scenario.

Recommendation

Consider doing a low level liquidity addition straight to the Trader Joe pair — this avoids the flawed implementation of the Uniswap router.

```
if (factory.getPair(wavaxAddress, tokenAddress) ==
address(0)) {
   pair = factory.createPair(wavaxAddress, tokenAddress);
}
WAVAX.deposit(wavaxReserve);
WAVAX.safeTransfer(pair, wavaxReserve);
token.safeTransfer(pair, tokenAllocated);
lpSupply = IUniswapV2Pair(pair).mint(address(this));
```

Resolution

RESOLVED

A low level addition is done.

Issue #04	LaunchEvent is a proxy but does not use upgradeable OpenZeppelin dependencies
Severity	LOW SEVERITY
Location	Line 18 `contract LaunchEvent is Ownable {`
Description	The LaunchEvent is a non-upgradeable proxy contract as it is deployed using the Clones library. This allows for the launch event code to be stored in a separate implementation contract from the factory and also reduces the deployment cost of launch events. However, given that this contract is a proxy, it should still use the upgradeable OpenZeppelin dependencies as these have `init` functions.
Recommendation	Consider consistently using upgradeable dependencies with Clones contracts. In fact, as Ownable is unused, it should simply be removed completely in this instance.
Resolution	RESOLVED The dependencies `SafeERC20Upgradeable` and `IERC20MetadataUpgradeable` have been imported and are used within the contract.

Issue #05	**Best practice: Doing the user transfer before the fee transfer on withdrawals increases the exploit risk in case the contract is modified**
Severity	LOW SEVERITY
Location	Line 370 `_safeTransferAVAX(rocketJoeFactory.penaltyCollector(), feeAmount);`
Description	The `withdrawAvax` function presently first sends the withdrawn AVAX to the user, after which the fee percentage is sent to the Trader Joe penalty receiver. This is less than ideal as the user can re-enter into other portions of the code before the final critical aspect of the function has been reached. Although innocent looking in the current iteration of the contract, if Trader Joe were to do away with storing the balance as WAVAX and instead used AVAX, a malicious party could call `skim()` at this point and cause the `skim` function to take out more AVAX then it should. This is because the contract balance will still be higher than the `avaxReserve` variable at this point in time.
Recommendation	First of all, consider inverting the order of the two transfers so that the less trusted transfer to the user occurs last. Secondly, consider protecting `skim` with an EOA-only requirement and `_safeTransferAVAX` with a requirement that at any point that it is called, there must still be enough AVAX within the contract to cover the `avaxReserve`.
Resolution	RESOLVED The two transfers are inverted, `_safeTransferAVAX` has been made EOA-only and has a strict `reserve` requirement. We commend Trader Joe for taking this issue seriously.

Issue #06	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains typographic errors on the following lines of code.
	
	Line 6
	`import “@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol”;`
	
	The contract can simply import IERC20 (or IERC20Upgradeable) as the metadata functionality is unused.
	
	Line 31
	`/// smaller than \`maxAllocation\``
	
	This comment should say smaller than or equal to.
	
	Line 203
	`/// @notice Initialise the launch event with needed paramaters`
	
	This notice is inconsistent with the 'initialize' function name.
	
	Line 211
	`/// @param _maxAllocation The maximum amount of AVAX depositable`
	
	This parameter documentation should likely indicate that the allocation is per-user (this misled our auditors for a moment while first reading the documentation).
	
	Line 518
	`/// @notice Stops the launch event and allows participants withdraw deposits`
	
	This comment should say "'to' withdraw deposits”.
	
	Line 559
	`uint256(PHASE_ONE_DURATION - PHASE_ONE_NO_FEE_DURATION);`
	
	This is an unnecessary cast to uint256, simple brackets would have sufficed.

Recommendation	Consider fixing the above typographical errors.
Resolution	RESOLVED

Issue #07	Lack of validation
Severity	INFORMATIONAL

Description	The contract contains sections of code which lack proper validation. This could cause errors in case unexpected inputs are provided.
	
	Line 257
	`issuer = _issuer;`
	
	Issuer should be explicitly validated to be non-zero.
	
	Line 276
	`floorPrice = _floorPrice;`
	
	The floor price should be explicitly validated to be non-zero as a zero floor price would break the business logic.
	
	Line 281
	`maxAllocation = _maxAllocation;`
	
	The `maxAllocation` should be explicitly validated to be non-zero as a zero maximum allocation would break the business logic.

Recommendation	Consider using the following requirements:
	
	`require(_issuer != address(0), "");`
	`require(_floorPrice > 0, "");`
	`require(_maxAllocation > 0, "");`

Resolution	RESOLVED The `issuer` and `maxAllocation` are now validated. `_floorPrice` was not validated but after discussing with the client, a zero `floorPrice` is in fact allowed and does not seem to cause side-effects.

Issue #08	`withdrawAVAX` can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the `external` keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the variable as external.
Resolution	RESOLVED

Issue #09	Lack of event for `withdrawIncentives`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Consider adding an event for the function.
Resolution	RESOLVED

Issue #10	**`getPenalty` reverts ambiguously if the launch has not started yet**
Severity	INFORMATIONAL
Location	Line 552 `uint256 timeElapsed = block.timestamp - auctionStart;`
Description	The `getPenalty` function underflows on the first line in case the `auctionStart` variable is still in the future. In this case, it is not immediately clear to a user or tool integrator what is happening since there is no reversion message.
Recommendation	Consider explicitly handling this case. In case the client chooses to revert explicitly within this code location it might make sense to also revert once phase 3 is reached.
Resolution	RESOLVED

2.3 RocketJoeStaking

The RocketJoeStaking contract is a simple staking contract inspired by the original SushiSwap MasterChef.

RocketJoeStaking allows for the staking of JOE tokens. Users that stake JOE tokens will receive rJoe in return over time. It is the only way for users to acquire Rocket Joe tokens and given that these tokens are non-transferable, it requires users to stake JOE instead of purchasing it to participate in a LaunchEvent.

The owner of the contract can select the mint rate of the contract, set in rJoe tokens per second. The staking contract contains no notion of deposit fees or lockups. Contrary to the SushiSwap MasterChef, there are also no individual pools to deposit into. Instead there is a single JOE pool.

The RocketJoeStaking contract is upgradable.

2.3.1 Privileges

The following functions are privileged functions that can be called by the owner:

- `updateEmissionRate`
- `transferOwnership`
- `renounceOwnership`

2.3.2 Issues & Recommendations

Issue #11	Governance risk: The staking contract is upgradeable
Severity	HIGH SEVERITY
Description	Currently, the RocketJoeStaking contract is upgradeable which means that the admin of the proxy contract can change the logic over time to anything including potentially logic that takes all staked tokens out of the contract without users' consent. The client should therefore be careful with their governance setup and ensure that the likelihood of such events is minimized as they might have catastrophic consequences for users. It should be noted that the client has already been managing upgradeable contracts successfully for Banker Joe for quite some time.
Recommendation	Consider safeguarding the admin role of the proxy to ensure that in case a key is stolen this does not have catastrophic consequences. Our recommendation is always to first have a validation period to ensure no irregularities occur where the admin key is simply the team multisig. After this initial validation period, a timelock can be inserted between the multisig and staking contract as a further safeguard.
Resolution	ACKNOWLEDGED The Trader Joe team acknowledges this issue and has indicated they will be carefully managing these contracts from a multi-sig. This issue can be marked as resolved or partially resolved once Paladin has confirmed that this is in fact the case on the deployed contracts and that the multi-sig participants are reputable.

Issue #12	Contract uses manipulatable contract token balance to figure out the total number of JOE staked
Severity	INFORMATIONAL
Location	Line 83 `uint256 joeSupply = joe.balanceOf(address(this));`
Description	To calculate the total supply in the staking contract, the contract uses `balanceOf` which simply counts the number of Joe tokens within the contract. As with most contracts that have this feature, it is inherited from the flawed original SushiSwap contract. Usage of `balanceOf` is discouraged to derive the `totalSupply` as it can be manipulated. In contrast, keeping track of a global `totalSupply` variable that is adjusted on `deposit`, `withdraw` and `emergencyWithdraw` is safer as it cannot be manipulated. The `balanceOf` method also has a higher gas cost and has caused severe exploits to projects that used it with a fee-on-transfer token (which does not apply for this audit as JOE does not have such taxes). It should generally be avoided.
Recommendation	Consider adding a `totalJoeStaked` variable and updating it on `deposit`, `withdraw` and `emergencyWithdraw`. This variable must then be consistently used in the `pending` and `update` functions.
Resolution	RESOLVED A `totalJoeStaked` variable was added and is incremented and decremented properly.

Issue #13	**Best practice: `deposit` does not adhere to checks-effects-interactions**
Severity	INFORMATIONAL
Location	Line 105 `_safeRJoeTransfer(msg.sender, pending);`
Description	The `deposit` function does an rJoe transfer within the middle of the function. This does not adhere to checks-effects-interactions and is a theoretical reentrancy vulnerability. This issue has been marked as informational as rJoe does not allow for reentrancy and there is therefore no way for a malicious user to exploit this. However, as Paladin believes that the checks-effects-interactions pattern is one of the most crucial Solidity patterns to follow, we are still raising this issue from an educative perspective in an effort to increase awareness amongst developers to more consistently adhere to it.
Recommendation	Consider initializing `pending` to zero and adding an `if (pending != 0)` section at the end of deposit (a non-zero check could be considered within `withdraw` as well.
Resolution	RESOLVED

Issue #14	**Check within updatePool that validates that nobody has staked yet can be circumvented by transferring a nominal JOE token to the contract leading to lost rewards**
Severity	INFORMATIONAL
Location	Line 163 `if (joeSupply == 0) {`
Description	The `joeSupply == 0` check can be circumvented by transferring 1 Joe in without a deposit. This causes some rJoe to remain unclaimable within the contract as the real number of deposits is still zero at this point, yet the contract starts minting rJoe. This issue is purely informational as it is simply a side-effect of a previously mentioned issue. It is marked as informational since the impact is so minimal.
Recommendation	Consider using a `totalJoeStaked` variable as was recommended in a previous issue.
Resolution	RESOLVED

Issue #15	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains typographical errors on the following lines of code.
	
	Line 106
	`user.amount = user.amount + _amount;`
	
	Within Solidity v0.8, `+=` can be used to simplify sections like this.
	
	Line 128
	`user.amount = user.amount - _amount;`
	
	Within Solidity v0.8, `-=` can be used to simplify sections like this.
	
	Line 144
	`joe.safeTransfer(address(msg.sender), _amount);`
	
	Wrapping `msg.sender` in `address` is not necessary. This has been done multiple times throughout the contract.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

Issue #16	**Lack of validation on `updateEmissionRate` could cause excessive minting**
Severity	INFORMATIONAL
Location	Line 151 `function updateEmissionRate(uint256 _rJoePerSec) external onlyOwner {`
Description	The contract contains a section of code which lacks proper validation. This could cause errors in case unexpected inputs are provided. Lack of a maximum on `_rJoePerSec` could cause `updatePool` to revert due to overflow and could furthermore cause excessive rewards if it was accidentally set too high.
Recommendation	Consider using the following requirement. `require(_rJoePerSec <= MAX_EMISSION_RATE, "");`
Resolution	RESOLVED

2.4 RocketJoeFactory

The Rocket Joe Factory is the main management contract that projects will use to create and deploy new launch events.

Trader Joe, which is the owner of this contract, can specify various properties that will be used for all launch events created by this contract's `createRJLaunchEvent` function. One of the global parameters is how much rJoe should be burned per AVAX, which is initially set to 100 rJoe per AVAX.

To deploy new launch event instances, the Rocket Joe platform uses the Clones pattern which is similar to deploying upgradable proxies without the upgradable aspect of them. This means that Rocket Joe deploys and verifies an implementation contract once and all LaunchEvent instances will use this. This implementation contract is called the eventImplementation and once launches are deployed, they cannot change or upgrade their implementation.

Through the Rocket Joe Factory, Trader Joe can create a Rocket Joe Launch Event. It should be noted that an event can only be created once for any given token and that there must not be any liquidity in the Trader Joe pair yet. Since a `createRJLaunchEvent` call must require the sender to transfer tokens, the chances of someone locking the actual team out (as a token can only be launched once) are slim as nobody is supposed to have tokens at this stage of the project.

2.4.1 Privileges

The following functions can be called by the owner:

- `setEventImplementation`
- `createRJLaunchEvent`
- `setPenaltyCollector`
- `setRouter`
- `setFactory`
- `setRJoePerAvax`
- `setPhaseDuration`
- `setPhaseOneNoFeeDuration`
- `transferOwnership`
- `renounceOwnership`

2.4.2 Issues & Recommendations

Issue #17	`eventImplementation` cannot be changed
Severity	LOW SEVERITY
Location	Line 20 `address public override eventImplementation;`
Description	The Rocket Joe factory is permanently set in the Rocket Joe token contract. The launch event template (`eventImplementation`) is also permanently set in the Rocket Joe Factory. This means that to add a feature to the launch event template, a whole new Rocket Joe Token instance would need to be created. Not being able to ever add new features to the event implementation might be a significant administrative drawback as Trader Joe might want to add new features over time. Doing so is presently impossible.
Recommendation	Consider whether Trader Joe ever wants to be able to upgrade or apply fixes to the `eventImplementation`. If so, consider adding an `onlyOwner` `setEventImplementation` function. As with all versioned software, the client should be careful if they call this function, as some old events might have an old implementation while more recent events have a new implementation. If the event interface is not compliant, third-party tools or contracts might stop working as they still expect the old interface.
Resolution	RESOLVED A `setEventImplementation` function was added.

Issue #18	`safeTransferFrom` should be used
Severity	LOW SEVERITY
Location	Line 132 `IERC20(_token).transferFrom(msg.sender, launchEvent, _tokenAmount);`
Description	`safeTransferFrom` is not used to transfer in the launch token from the project creator. This could cause certain malformed tokens to not be eligible for a launch event.
Recommendation	Consider consistently using `safeTransferFrom`.
Resolution	RESOLVED

Issue #19	`eventImplementation` and `wavax` can be made `immutable`
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variables explicitly immutable. It should be noted that if the `eventImplementation` is made settable, it can no longer be made immutable.
Resolution	RESOLVED Since the client has added a function to change the `eventImplementation`, there is no need for this variable to be `immutable`. This contract has also been made upgradable.

Issue #20	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains typographic errors on the following lines of code.
	
	Line 59
	`"RJFactory: Addresses can't be null address”`
	
	This comment should indicate the zero address, as null values are not possible within solidity.
	
	Line 127
	`"RJFactory: liquid pair already exists"`
	
	Change to: `liquidity pair already exists`
	
	Line 134
	`ILaunchEvent(payable(launchEvent)).initialize(`
	
	Casting launchEvent to payable is unnecessary.
	
	Line 207
	`"RJFactory: phase one duration lower than no fee duration”`
	
	This exception should say lower than or equal.
	
	Line 224
	`"RJFactory: no fee duration bigger than phase one duration”`
	
	This exception should say bigger or equal.

Recommendation	Consider fixing the above typographical errors.
Resolution	RESOLVED

Issue #21	**`maxAllocation` is vulnerable to Sybil attacks**
Severity	INFORMATIONAL
Location	Line 106 `uint256 _maxAllocation,`
Description	The `maxAllocation` value indicates the maximum allocation per user. There is however nothing stopping a user from creating many wallets to participate with and still get a large allocation.
Recommendation	As Sybil resistance is an extremely difficult topic to solve, we have no easy recommendation. We have seen well-known launchpads utilize KYC procedures to do this but expect this to not match with the ethos of Rocket Joe.
Resolution	ACKNOWLEDGED The client agrees that there is no easy solution and they do not wish to burden users with KYC requirements.

Issue #22	**Setting rJoe back to a previous rJoe token will be impossible since it is already initialized**
Severity	INFORMATIONAL
Location	Line 159 `IRocketJoeToken(_rJoe).initialize();`
Description	The contract contains logic to update the rJoe token instance. In our personal opinion, this logic is not exactly necessary as a new factory can be deployed if a new Rocket Joe token was required. However, if the governance wants to switch to a new token instance but then decide that they want to switch back, this will not be possible since the `setRJoe` function always calls `initialize`.
Recommendation	Consider either removing the `setRJoe` function completely or making initialization optional.
Resolution	RESOLVED The `setRJoe` function was removed completely.

Issue #23	Lack of validation
Severity	INFORMATIONAL
Location	Line 166 `function setPenaltyCollector(address _penaltyCollector)`
Description	The contract contains a section of code which lacks proper validation. This could cause errors in case unexpected inputs are provided.
Recommendation	Consider using the following requirement. `require(_penaltyCollector != address(0));`
Resolution	RESOLVED

Issue #24	Lack of events for `setPhaseDuration` and `setPhaseOneNoFeeDuration`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Consider adding events for the above functions.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY